CS230

Changing people's hair color in images

Clemens Macho
Department of Computer Science
Stanford University
cmacho@stanford.edu

Figure 1: a photo of the author transformed into various colors using our algorithm haircolorGAN

Abstract

Many popular apps provide functionality for changing the hair color of people in images. We design and implement a variation of the cycleGAN architecture that can learn this transformation from a dataset of real images with hair color labels, without being shown input-output pairs. In contrast to cycleGAN, our architecture uses only one generator and one discriminator, and both of them take hair color labels as input in addition to images.

1 Introduction

The goal of my project was to build a system that is able to change the hair color of people in images. To be more precise, it takes a picture of a person and a target hair color as input and outputs the same picture but with the hair color changed to the target hair color. In addition, the system should learn this transformation on its own from a dataset of natural photos of people with various hair colors, without being shown input-output pairs.

In recent years, generative adversarial networks (GANs) have achieved great results in similar problems of image-to-image translation. We use a variation of the popular cycleGAN architecture [1]. One limitation of cycleGAN is that it is built to translate between two domains X and Y, but we want to be able to match a wider range of hair colors. Instead of using two generators, we use only

CS230: Deep Learning, Winter 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

one generator and one discriminator, but they both receive color labels (RGB values) as additional input in order to determine what hair color the generated image should have. Our GAN architecture (haircolorGAN) is shown in figure 2. It will be described in more detail in section 4.

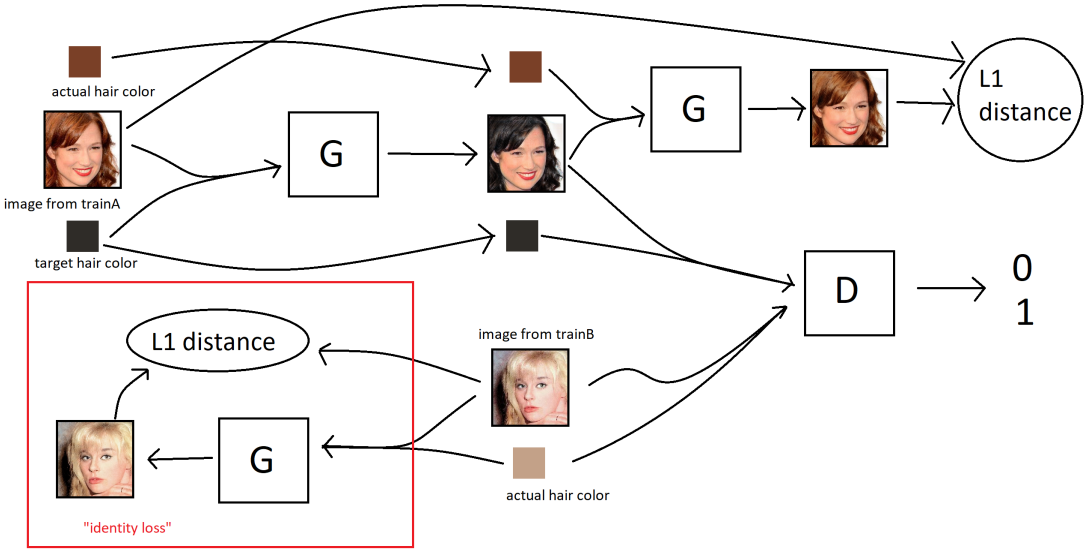

Figure 2: our proposed haircolorGAN architecture for translating between multiple hair colors

2 Related work

The most relevant paper is [1], already mentioned in the previous section. In contrast to pix2pix [2], which is a GAN algorithm for image-to-image translation that learns from a data set of input-output pairs, the algorithm in [1] can learn from unpaired data. The same method was discovered independently in [3]. The authors present hair color transformation between two hair colors, blonde and black, as an example application.

In starGAN [4] a variation of the cycleGAN architecture is proposed that can translate between multiple domains. I am using some ideas from this paper such as using only one generator and one discriminator and passing both images and labels as input to the generator. There are some differences between the architecture described in [4] and the one described in figure 2, e.g. the input/output behavior of the discriminator. In this respect, my approach is closer to [1]. Another difference is that in [4] they use discrete labels, whereas the labels in my model (average RGB value of the hair in the picture) can represent a continuous range of colors.

3 Datasets

I used the CelebAMask-HQ dataset [5], which contains 30.000 images of celebrities and comes with masks that show where the hair is.

Figure 3: examples from the celebAMaskHQ dataset

I used these masks in order to compute RGB values for the hair color in each of the images. The RGB value was computed as the average over all pixels that belong to the person's hair in the image. I also used the masks in order to filter out images where the hair covers less than 7.5% of the image.

4 Methods

We are now going to describe the training procedure as sketched in figure 2 in more detail. The training set is split into two sets trainA and trainB. The images from trainB are presented to the discriminator with their actual hair color. Images from trainA are given as input to the generator along with a target hair color that is randomly sampled from all the hair colors occuring in trainB. The discriminator tries to classify images from trainB (labeled with their actual hair color) as 1 and generated images (labeled with the target hair color) as 0, whereas the generator tries to fool the discriminator with realistic generated images. In order to be successful at this, the generator should have to match the target hair color in the generated image.

We use mean squared error for the GAN-loss. Let us denote images by x, y and hair color labels by h. Then the objective of the discriminator can be described as

$$\mathcal{L}_D = \frac{1}{2}\mathbb{E}_{(y,h)\in\text{trainB}}\left[(D(y,h)-1)^2\right] + \frac{1}{2}\mathbb{E}_{(x,h_1)\in\text{trainA},(y,h_2)\in\text{trainB}}\left[(D(G(x,h_2),h_2)-0)^2\right].$$

Note that trainA and trainB come from the same distribution so the notation could be simplified slightly by dropping the distinction between them. We are going to continue treating them as distinct throughout this section. The corresponding component of the generator's objective function is

$$\mathcal{L}_{G,\text{GAN}} = \mathbb{E}_{(x,h_1)\in\text{trainA},(y,h_2)\in\text{trainB}}\left[(D(G(x,h_2),h_2)-1)^2\right].$$

In order to complete the cycle as in cycleGAN, generated images are fed back to the generator with the original hair color as target hair color and the output should be close to the original image.

$$\mathcal{L}_{G,\text{cycle}} = \mathbb{E}_{(x,h_1)\in\text{trainA},(y,h_2)\in\text{trainB}}\left[\|G(G(x,h_2),h_1)-x\|_1\right].$$

The "identity loss" as shown in the red square in figure 2 is less important than the rest of the diagram, but it is an additional way of training the generator in order to ensure that generated images remain similar to the original image. This was described in the section about "Photo generation from paintings" in [1]. It is sometimes used in the cycleGAN algorithm, but not always.

$$\mathcal{L}_{G,\text{identity}} = \mathbb{E}_{(y,h)\in\text{trainB}}\left[\|G(y,h)-y\|_1\right].$$

The total loss for the generator is then

$$\mathcal{L}_G = \mathcal{L}_{G,\text{GAN}} + \lambda_{\text{cycle}}\left(\mathcal{L}_{G,\text{cycle}} + \lambda_{\text{identity}}\mathcal{L}_{G,\text{identity}}\right),$$

where $\lambda_{\text{identity}}$ and λ_{cycle} are hyperparameters. These hyperparameters are quite important: When they are large, the generator will be more reluctant to make changes to the hair color or other parts of the image because it is incentivized to keep images the same.

The above notation suggests that (x, h_1) and (y, h_2) are sampled uniformly and independently from trainA and trainB. In practice, this can lead to a large percentage of training examples where h_1 is very similar to h_2 (e.g. both are dark brown) so that the generator barely has to do any work and may not get enough interesting examples to learn from. Therefore I also experimented with sampling from the training sets according to the following procedure. In order to sample one training example $((x, h_1), h_2)$:

1. Sample K examples from trainA: $C_1 = [(x_1, h_{1,1}), \ldots, (x_K, h_{1,K})]$.
2. Sample K examples from trainB: $C_2 = [(y_1, h_{2,1}), \ldots, (y_K, h_{2,K})]$.
3. Shuffle C_2 repeatedly (L times). Keep whichever ordering maximizes $\sum_{i=1}^{K} \|h_{1,i} - h_{2,i}\|_2$.
4. Pick a random index $i \in \{1, \ldots, K\}$ and set $(x, h_1) \coloneqq (x_i, h_{1,i})$ and $(y, h_2) \coloneqq (y_i, h_{2,i})$.

Here, K and L are hyperparameters. This algorithm results in a greater expected difference $\|h_{1,i} - h_{2,i}\|_2$ without changing the marginal distributions for (x, h_1) and (y, h_2). In other words, every $(x, h_1) \in$ trainA is equally likely to be picked by this procedure and similarly every $(y, h_2) \in$ trainB is equally likely to be picked by this procedure. Step 3 above is a Monte Carlo algorithm for maximizing $\sum_{i=1}^{K} \|h_{1,i} - h_{2,i}\|_2$. I experimented with several values for K (between 1 and 20) and L (we can use 1000 for L without slowing down the training procedure).

5 Neural network architecture

I initially used the same neural network architectures that were used in [1], with the difference that I used 6 channels for the input: 3 channels for the image and 3 channels for the hair color label.

For the generator they use a residual neural network. It consists of two stride-two convolutions, then a "bottle-neck" of several residual blocks and then two deconvolutional layers to get back to the original height and width.

For the discriminator, they use the so called PatchGAN architecture. It does not output one prediction but a 30x30 grid of predictions for overlapping image patches. A convolutional neural network architecture is used in order to reuse lower level features for predictions on overlapping image patches. The loss is then calculated separately for the prediction on each image patch and the resulting losses are averaged in order to get a total loss.

Later I wanted to try whether a discriminator that takes the entire composition of the image into consideration could lead to higher quality results than PatchGAN. I added some linear layers and a single output layer with a sigmoid nonlinearity at the end of the PatchGAN architecture in order to get one prediction for the entire image instead of predictions for image patches. Note that LSGAN loss [6] is usually not used with a sigmoid nonlinearity at the output layer. The sigmoid nonlinearity could potentially lead to vanishing gradient problems in GANs. It turned out to work similarly well as the PatchGAN architecture for this problem.

6 Normalization layers

During my first attempts at training the model, I observed that the generator didn't seem to take the hair color labels into account at all. When I used large values for the parameters K and L described at the end of section 4, the generator did change the hair color but it changed it to seemingly random colors. When I used $L = K = 1$, the generator didn't change the hair color at all. After much analysis, I found that the problem was due to the instance normalization layers [7] that were used in cycleGAN and that I had been using. Since the input channels for the hair color labels were constant across the width and height dimensions, they were completely zeroed out by the first instance normalization layer in the generator, even though there was a (linear) convolutional layer before that. The later layers had no access to any information about the input label. For the discriminator, the information from the color label was not completely wiped out because there was a nonlinearity before the first normalization layer. In the case when I used large values for L and K, the generator was apparently learning to predict the target color based on the hair color in the input image, since these were correlated.

After I replaced the instance normalization layers by batchnorm layers and increased the batch size to 8, the generator started taking the input labels into account. For the Sigmoid Discriminator described at the end of the previous section, I omitted normalization layers entirely.

7 Results

I trained several versions of the model with different choices of hyperparameters and neural network architectures. Different versions seem to perform better or worse depending on the input, but some trends can be seen. See figure 4 for a comparison. Note that for each training session, the networks were trained for a total of 26 epochs of 12.000 iterations each, with the learning rate decaying linearly for the last 12 epochs. In figure 4 we can see that the model that uses the Sigmoid Discriminator seems to change the hair color less in epoch 24. We see the same phenomenon on other input images. This may be due to a vanishing gradient problem because the learning rate is very low in the last

epochs. The gradients coming from the discriminator vanish while the gradients from the identity loss still reach the generator, leading to output images that are more similar to the input. Similarly, lower values of $\lambda_{\text{identity}}$ seem to lead to stronger changes in the hair color. A direct comparison of the PatchGAN and Sigmoid discriminators cannot be drawn from these experiments since different hyperparameters were used.

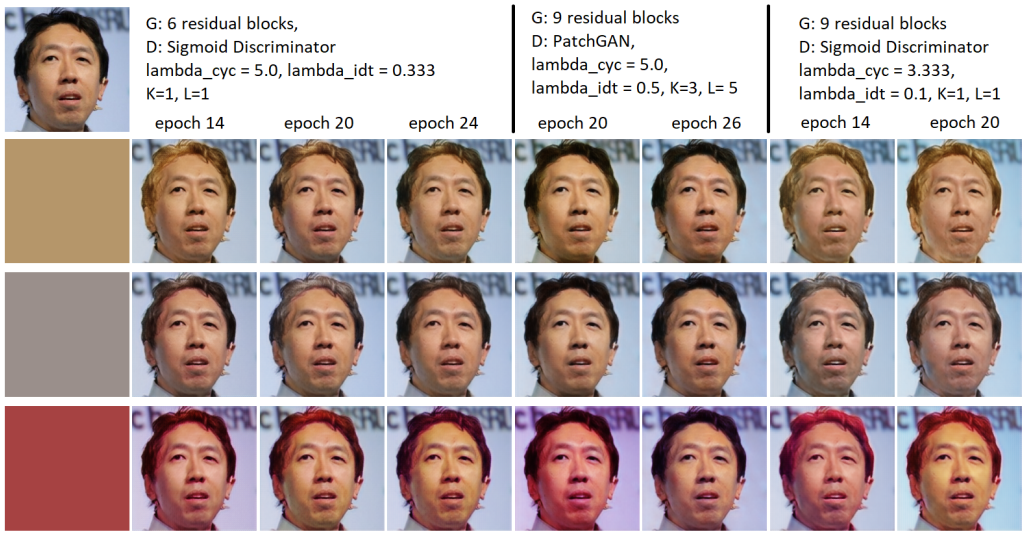

Figure 4: different versions of the model applied to an image of Prof. Ng

Unfortunately, each of the trained models sometimes fails completely or almost completely to change the hair color. To give a sense of how well it works or doesn't work, figure 5 shows six uncurated examples generated with the version that uses the Sigmoid Discriminator and $\lambda_{\text{identity}} = 0.333$ and has been trained for 20 epochs (third column in figure 4). The examples that I have underlined in red in figure 5 are ones where the transformation seems to have failed completely. For the one underlined in blue, no change was necessary, because the target hair color already matched the image.

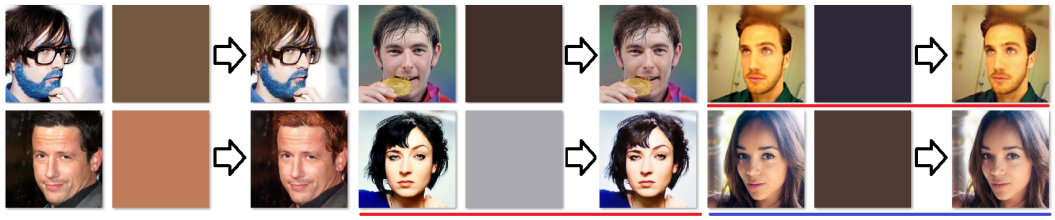

Figure 5: six uncurated examples. The input images and colors were sampled from a hold-out test set of approx. 2.000 images according to the procedure described in section 4 with $K = 20, L = 20.000$.

To give a more quantitative sense, I counted for 100 such test examples, how often the transformation worked and how often it failed. For 55 of them, the transformation worked reasonably well, for 6 it changed only parts of the hair, for 9 no change was necessary because the hair color already matched the target hair color and for 30 the transformation failed completely.

8 Conclusion

I have demonstrated that my haircolorGAN architecture works, even though I couldn't get it to work reliably. While I have identified some hyperparameters that seem to play an important role, further experiments and more analysis would be necessary to determine which hyperparameters have the most significant impact on the outcome and what modifications could bring about the greatest improvement.

References

[1] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

[2] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[3] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. *CoRR*, abs/1703.05192, 2017.

[4] Yunjey Choi, Min-Je Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. *CoRR*, abs/1711.09020, 2017.

[5] Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. *CoRR*, abs/1907.11922, 2019.

[6] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Multi-class generative adversarial networks with the L2 loss function. *CoRR*, abs/1611.04076, 2016.

[7] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.